Optimizing the Minimum Vertex Guard Set on Simple Polygons via a Genetic Algorithm

ANTONIO L. BAJUELOS [1], SANTIAGO CANALES [2], GREGORIO HERNÁNDEZ [3],
ANA MAFALDA MARTINS [1]

[1] Department of Mathematics & CEOC
University of Aveiro
Campus de Santiago, 3810-193, Aveiro
PORTUGAL
{leslie,mafalda.martins}@ua.pt

[2] Escuela Técnica Superior de Ingenieria
Universidad Pontifícia Comillas de Madrid
C/ Alberto Aguilera 21, Madrid
SPAIN
scanales@upcomillas.es

[3] Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, Boadilla del Monte, 28660 Madrid
SPAIN
gregorio@fi.upm.es

Abstract: - The problem of minimizing the number of vertex-guards necessary to cover a given simple polygon (MINIMUM VERTEX GUARD (MVG) problem) is NP-hard. This computational complexity opens two lines of investigation: the development of algorithms that establish approximate solutions and the determination of optimal solutions for special classes of simple polygons. In this paper we follow the first line of investigation and propose an approximation algorithm based on general metaheuristic genetic algorithms to solve the MVG problem. Based on our algorithm, we conclude that on average the minimum number of vertex-guards needed to cover an arbitrary and an orthogonal polygon with n vertices is $n/6.38$ and $n/6.40$, respectively. We also conclude that this result is very satisfactory in the sense that it is always close to optimal (with an approximation ratio of 2, for arbitrary polygons; and with an approximation ratio of 1.9, for orthogonal polygons).

Key-Words: Computational Geometry, Art Gallery Problems, Visibility, Approximation Algorithms, Metaheuristics, Genetic Algorithms

1 Introduction

The Art Gallery Problems are well-studied visibility problems in Computational Geometry [20]. The *original* problem was introduced by Victor Klee, in 1973, when he proposed the following problem: *How many stationary guards are needed to cover an art gallery room with n walls*? Informally the floor plan of the art gallery room is modeled by a *simple polygon Q* (simple closed polygon with its interior). A guard is considered to be a fixed point in Q with visibility range 2π. We say that a point x *sees* point y (or y is visible to x) if the line segment connecting both does not intersect the exterior of Q. A set of guards covers Q, if each point of Q is visible by at least one guard. Thus, the art gallery problem deals with setting a minimal number of guards in a gallery room whose floor plan has a polygonal shape, so that they can see every point in the room. Two years later Chvátal established the well known *Art Gallery Theorem*: $\lfloor n/3 \rfloor$ *guards are occasionally necessary and always sufficient to cover a simple polygon of n vertices* [9]. Over the years, numerous variations of the original problem have been considered and studied, such as: location of the guards (anywhere or

in specific positions, e.g., on vertices), different types of guards (e.g., stationary guards versus mobile guards) and different assumptions on the input polygon (e.g., orthogonal simple polygons, i.e., simple polygons whose edges meet at right angles), see [20]. An interesting variant is the *Orthogonal Art Gallery Theorem*. This theorem was first formulated and proved by Kahn et al. [13], in 1983. It states that $\lfloor n/4 \rfloor$ *guards are occasionally necessary and always sufficient to cover an orthogonal simple polygon of n vertices*. Orthogonal simple polygons are an important subclass of polygons. As a matter of fact, they are useful as approximations to polygons and they arise naturally in domains such as raster graphics, VLSI design or architecture. Efficient algorithms, based on the proofs of the above theorems, were developed to guard both arbitrary and orthogonal simple polygons with $\lfloor n/3 \rfloor$ and $\lfloor n/4 \rfloor$ guards, respectively. Although these numbers are necessary in some cases, they often exceed the number of guards needed to cover a particular simple polygon. A variant of art gallery problem is the MINIMUM VERTEX GUARD (MVG) problem, which is the problem of finding the minimum number of guards placed on vertices (*vertex-guards*) needed to cover a given simple polygon. This is a NP-hard problem both for arbitrary and orthogonal simple polygons [14, 18].

Our contribution. The computational complexity of the MVG problem for simple polygons opens two lines of investigation: the development of algorithms that establish approximate solutions and the determination of optimal solutions for special classes of simple polygons (e.g., [5]). In this paper we follow the first line of investigation. We propose an approximation algorithm based on general metaheuristic genetic algorithms to solve the MVG problem on simple polygons (arbitrary and orthogonal). Since the optimal solution to the MVG problem is unknown, we use a method that allows us to determine a *lower bound* for our algorithm, as in [4]. This way, we are able to find the approximation ratio of our technique. Our implementation was developed using the CGAL library [8] and our experiments were performed on a large set of randomly generated simple polygons.

This paper is structured as follows: in the next section we introduce some preliminary definitions and useful results. In section 3 we present a strategy, based on general metaheuristic genetic algorithms, to solve the MVG problem on simple polygons. In section 4 we establish greedy constructive algorithms that allow us to determine a lower bound for our algorithm. Section 5 is devoted to present our experiments and results on arbitrary and orthogonal polygons, subsections 5.1 and 5.2, respectively. Finally, in section 6 we draw conclusions and future work.

2 Preliminaries

As stated before, the MVG problem is NP-hard for simple polygons and a way to deal with this computational complexity is to develop approximation algorithms to tackle the problem. In general, these algorithms can be designed specifically to solve the problem (e.g., greedy constructive strategies) or they can be based on general metaheuristics. A metaheuristic is a general algorithmic framework which can be adapted to different optimization problems with minor adjustments (see [7, 11] for a comprehensive survey on the subject). There are several works where approximation algorithms (heuristics) were developed to solve the MINIMUM SET GUARD (MSG) problem (e.g., [4, 10, 15, 19]). In recent works, metaheuristic techniques have proven to be very well behaved in solving the MSG problem [3], as well as the MAXIMUM HIDDEN VERTEX SET problem [6], which is also a NP-hard visibility problem. On the other hand, genetic algorithms (GA) are very common metaheuristic techniques in computer science. They are extensively used to find approximate solutions of combinatorial optimization problems (e.g., [16, 21]).

For the above reasons, in this paper we propose an algorithm based on the metaheuristic GA to compute a small vertex-guard set for a given simple polygon *Q*.

Let *Q* be a simple polygon with *n* vertices $v_0, v_1, ..., v_{n-1}$ (we only study simple polygons in this paper, so we use the term polygon to refer a simple polygon). A vertex of *Q* is reflex if the interior angle between its two incident edges is greater than π, otherwise it is convex. We use *r* to represent the number of reflex vertices of *Q*. Without loss of generality, we assume that the vertices of *Q* are ordered in counterclockwise direction around the interior of *Q*. For a point $p \in Q$, we call *visibility polygon* of *p*, $Vis(p)$, to the set of all points $q \in Q$ that are visible to p, i.e., $Vis(p) = \{q \in Q : p \text{ sees } q\}$. We say that *G*, a given subset of vertices of *Q*, is a *vertex-guard set* of *Q* if *G* cover *Q*, i.e., if $\bigcup_{v \in G} Vis(v) = Q$. The cardinality of a vertex-guard set is denoted by $|G|$.

As the optimal solution for the MVG problem is unknown, we may ask: *How can we expect to prove that our approximate solutions are close to the real ones*? In order to answer this question, we developed a greedy constructive algorithm to calculate a large *visibility-independent set* on a given polygon. This gives us a method to compute a lower bound on the optimal number of vertex-guards for each instance of our experiments. Applying the approximation algorithm together with the algorithm to determine the lower bound, for each instance of our experiments, we get the performance ratio of our approximation algorithm.

In [4] a visibility-independent set of Q is defined as a finite set S, $S \subset Q$, such that $\forall p,q \in S,\ Vis(p) \cap Vis(q) = \phi$. The cardinality of a visibility-independent set is denoted by $|S|$. The example given in Fig. 1 illustrates a visibility-independent set of cardinality 3.

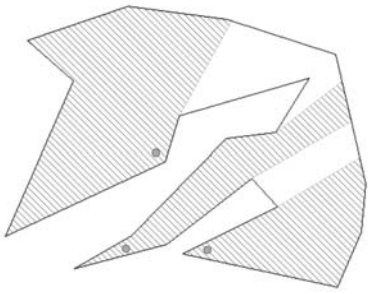

Fig.1: Visibility-independent set of an arbitrary polygon. Gray dots represent visibility-independent points.

t is easy to verify that no single vertex-guard is able to see more than one point of *S*, consequently $\forall G,S,\ |G| \geq |S|$. So the number of vertex-guards of a minimum-cardinality vertex-guard set of Q is *at least* the number of points in a maximum-cardinality visibility-independent set. Thus, the number of points on a maximum-cardinality visibility-independent set is a lower bound for the optimal number of vertex-guards on Q. Nevertheless, the problem of determining this lower bound is also NP-hard [4], so we developed approximation algorithms to tackle it, as already stated.

In our experiments, given a simple polygon, the main objective is to find a small vertex-guard set, *G*, and large visibility-independent set, *S*. The obtained set *G* approximates the optimal number of vertex-guards with an approximation ratio of $|G|/|S|$.

3 Approximation Algorithm Based on Genetic Algorithms (GA)

Genetic algorithms are a particular class of *evolutionary algorithms* that use techniques inspired by evolutionary biology such as *inheritance*, *mutation*, *selection*, and *crossover*/*recombination* (e.g., [1]). They are implemented as a computer simulation in which a population of abstract representations of candidate solutions (called *individuals* or *chromosomes*) to an optimization problem evolves toward better solutions. The evolution usually starts from an initial population of randomly generated individuals. These individuals are evaluated with a function (*fitness*) that indicates the adaptation degree of the individual to the environment. From this initial situation a series of iterations are realized in each of which there is simulated the creation of a new generation of individuals from the previous generation. This process consists of applying the genetic operators selection, crossover and mutation on the individuals. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. The final population, if the algorithm converges properly, will be composed of good individuals, the best of these being the solution given by the algorithm (see Fig.2).

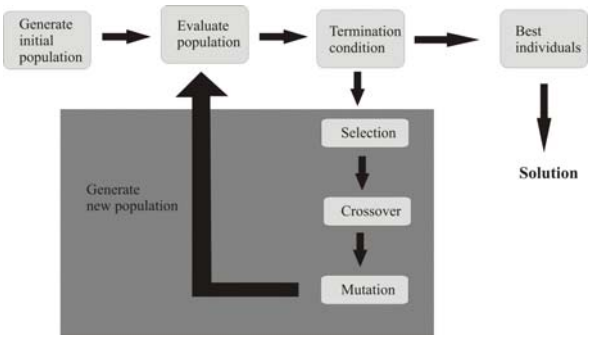

Fig.2: General scheme of a Genetic Algorithm.

The pseudo-code of a genetic algorithm follows:

Algorithm Genetic Algorithm

1. $t \leftarrow 0$
2. Initialize the population, $P(t)$
3. **while** termination condition not met **do**
4. Evaluate $P(t)$
5. Selection on $P(t)$
6. Recombine and/or Mutate $P(t)$
7. $t \leftarrow t+1$
8. Generate $P(t)$ from $P(t-1)$
9. **end while**

Taking into account the established above, to solve an optimization problem with the GA metaheuristic it is necessary to specify the following:

- a genetic representation of the possible solutions, called *individuals* or *chromosomes*, to the problem (*Encoding*);
- a way of creating an initial population of possible solutions (*Initial population*);
- a function to evaluate the individuals and simulate the process of natural selection, sorting solutions according to their "strength" (*Objective or Fitness function*);
- genetic operators to alter the composition of the solutions (*Selection*, *Crossover* and *Mutation*) and
- the values of various parameters used by the genetic algorithm (e.g., size of the population, probability of the genetic operators, evaluation of the population, generation of the population, termination condition).

In the next subsection we describe these items so they suit our problem.

3.1 GA's Parameters Definition

3.1.1 Encoding.

In our algorithm an individual (or chromosome) I is represented by a chain of 0's and 1's of length n, i.e., $I = g_0 g_1 ... g_{n-1}$, where each element g_i is called a *gene*. Each gene represents a vertex of the polygon, i.e., the i^{th} gene represents the vertex $v_i \in Q$. The value of each gene is 0 or 1. If the $g_i = 1$ then vertex v_i is an vertex-guard; otherwise ($g_i = 0$) vertex v_i is not a vertex-guard (see Fig.3).

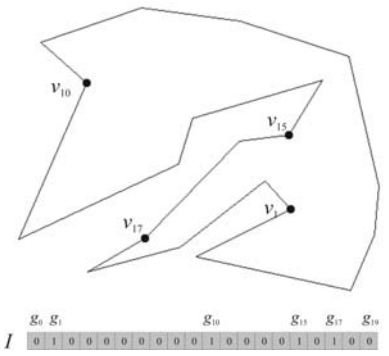

Fig.3: An individual I (for an arbitrary polygon with $n = 20$) and its representation. Black dots represent vertex-guards.

3.1.2 Initial Population

The population of a given generation/iteration consists of a set of individuals. The total number of individuals in each population has to be large enough to ensure diversity, but not too much as it worsens the algorithm's efficiency. In our case, we chose the population size to be the number of reflex vertices of the polygon connecting this way the input data of the problem and the elements of the metaheuristic. Thus, the population of the generation t in our algorithm is represented by: $P(t) = \{I_0^t, I_1^t, ..., I_{r-1}^t\}$, where each I_i^t represents an individual belonging to the population $P(t)$ and r is the number of reflex vertices of the polygon Q. Remember that an individual represents a possible solution for our problem, i.e., each individual must be a vertex-guard set. It has been proven that being Q a polygon with r reflex vertices, r guards placed on the reflex vertices of Q are always sufficient and occasionally necessary to guard Q [20]. Thus, in our algorithm, let $R = \{u_0, u_1, ..., u_{r-1}\}$ be the set of reflex vertices of Q. To create the initial population, $P(0)$, we generate each of the r individuals in the following way: $\forall i \in \{0,1,...,r-1\}$, if $R \setminus \{u_i\}$ is a vertex-guard set we mark all the vertices on $R \setminus \{u_i\}$ as vertex-guards; otherwise we mark all the vertices on R as vertex-guards. For example, in Table 1 we present the initial population, $P(0)$, of the polygon shown in Fig.4.

$I_0^0 = 01000000001001110100$
$I_1^0 = 11000000001001110100$
$I_2^0 = 11000000000001110100$
$I_3^0 = 11000000001000110100$
$I_4^0 = 11000000001001010100$
$I_5^0 = 11000000001001100100$
$I_6^0 = 11000000001001110000$

Table 1: Individuals of $P(0)$.

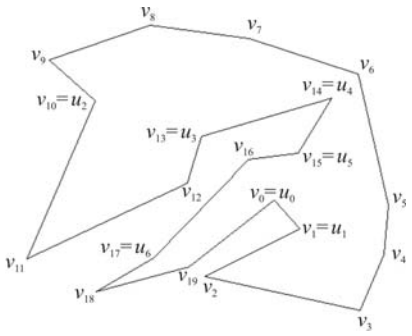

Fig.4: Polygon with $n = 20$, $r = 7$.

3.1.3 Objective or Fitness Function

The fitness function should help us to make the best selection of individuals to be reproduced. For that purpose, it assigns lower values to the solutions that are closer to the optimal. In our case, given an individual *I*, the fitness function returns the number of 1's that exist in the chain that represents it, i.e., $f(I) = \sum_{j=0}^{n-1} g_j$.

3.1.4 Selection

The selection method should choose the best individuals to be reproduced. Since there are many different types of selection, we performed a comparative study taking two common methods into account: the *roulette wheel selection* and the *tournament selection* (see, e.g. [17]). In the first method, the individuals are given a probability of being selected that is inversely proportional to their fitness. Two individuals are then randomly chosen, based on these probabilities, to be parents in crossover. In our case, we use this method to choose the two best individuals to be parents in crossover. In the tournament selection, *k* individuals are randomly selected and the best one is chosen for parenthood. We use a binary approach ($k = 2$), i.e., we select two pairs of individuals and choose as parents the lowest fitness value in each pair.

3.1.5 Crossover

Crossover operates on selected genes from parent individuals and creates new individuals (children). As there are many different kinds of crossover, we have done a comparative study with four different types of crossover: *single point crossover*, *two-point crossover*, *uniform crossover* and a variant of the single point crossover where the generated children cannot be clones of the parents (see, e.g. [17]). In any crossover method we only generate one child from two parents.

In single point crossover, a randomly selected point (gene) of the two parents is chosen and the parents are split at that crossover point. Finally, a child is created by exchanging its parents' tails (see Fig.5 (a)). In two-point crossover, two points of the parents are randomly selected and the fragment between the two points is exchanged with the corresponding fragment of the second individual (see Fig.5 (b)). Uniform crossover is an operator that decides (with some probability) which parent will contribute to each of the gene values of the child chromosomes. This allows the parent chromosomes to be mixed at the gene level rather than the segment level (as in the case of single and two-point crossover). If the probability is 0.5, approximately half of the genes of the child are inherited from one parent and the other half from the other. Fig.5 (c) illustrates a possible child after a uniform crossover. Finally, the last crossover operation is a variation of the single point crossover, where only positions that do not generate copies of the parents are allowed to be crossover points.

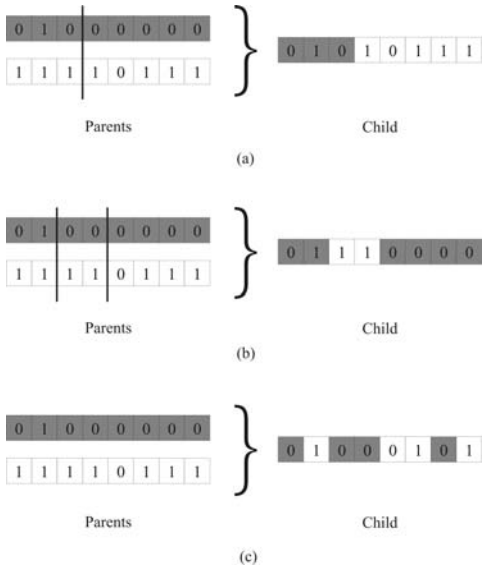

Fig.5: (a) Single point crossover; (b) Two-point crossover and (c) Uniform crossover.

Crossover only occurs with a given probability, p_c. The value of p_c is decided on the basis of trial and error, but it is generally between 0.7 and 0.95. We use $p_c = 0.8$. Note that the child resulting from any of the described crossover methods may not be valid (i.e., it may not correspond to a guard-vertex set), see Fig.6, in this case the child is not accepted.

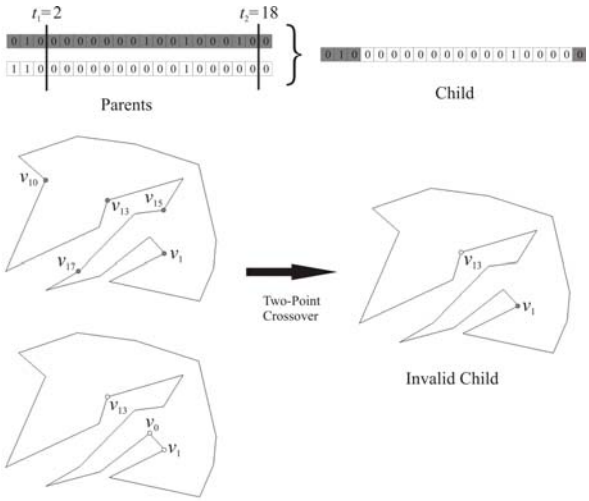

Fig.6: Two-Point crossover and an invalid child.

In this example, the obtained child is not valid because the polygon is not covered by the vertices v_1 and v_3.

3.1.6 Mutation

Since we use a binary encoding, the action of our mutation operation is relatively simple. For each binary digit (gene) it merely flips it from zero to one or vice versa, with a mutation probability, p_m (see Fig. 7).

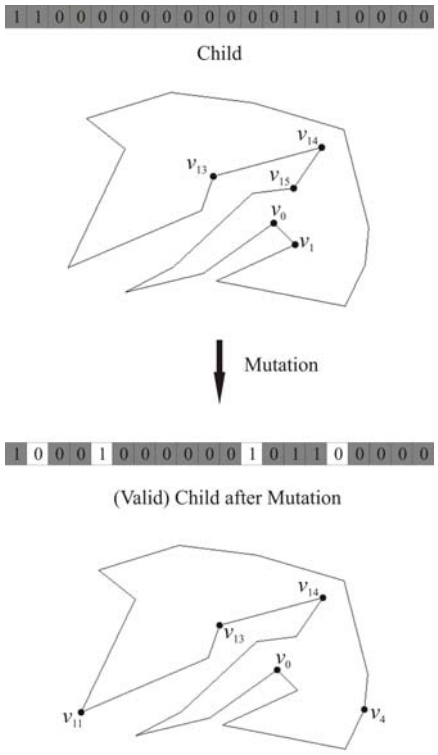

Fig.7: Mutation.

That probability is decided on the basis of trial and error, however it should be relatively low. In our case we apply the mutation to the child obtained in the crossover operation, with $p_m = 0.05$. As in the crossover, if the resultant individual is not valid we do not accept it.

3.1.7 Population's Generation

As there are many different ways to generate a new population we used a common one: we select the worst individual of the population to be deleted replacing it by the child obtained at the crossover. (see, e.g. [17]).

3.1.8 Population's Evaluation

We consider the evaluation of a population, i.e., the fitness of a population, $F(P(t))$, as the minimum value of the objective function when applied to all individuals of the population that is $F(P(t)) = \min\{f(I_0^t), f(I_1^t), ..., f(I_{r-1}^t)\}$.

3.1.9 Termination Condition

If in a sufficiently large number of generations the fitness does not change, then we can assume that we are close to the optimum. Thus, we terminate if the fitness of the population $F(P(t))$ remains unchanged for a given number of generations *h*. In our case, we consider $h = 500$ (this value has been empirically chosen).

3.2 Removing Redundant Vertex-Guards

After defining the GA's components, we obtain an approximation algorithm that allows us to obtain a vertex-guard set, *G*. However, it may be possible that exists a set $U \subset G$ such that $\bigcup_{v \in G \setminus U} Vis(v) = Q$. So, we apply a post-processing step where we iteratively remove those guards. This post-processing is done in the following way: for each $v_i \in G$, if *Q* is still covered by *G* without v_i, then v_i is removed from *G*; otherwise it remains as part of the set *G*.

4 Greedy Constructive Algorithms for Visibility-Independent Sets

As we described in section 2, visibility-independent sets provide us a method to compute a lower bound on the optimal number of vertex-guards. A natural approach to find a visibility-independent set *S* is to do so in a constructive greedy way: start with a set of candidates *C* (not visibility-independent), add visibility-independent points one by one to a set *S* (initially empty) selecting at each step a point from the candidate set *C*, according to some rule.

In our greedy algorithms we use the candidate set proposed by [4], which is $C = C_1 \cup C_2$, where C_1 denotes the convex vertices of *Q* and C_2 denotes the midpoints of the edges incident to two reflex vertices. Concerning the rule to select the points, we apply three different alternatives which results in three different greedy algorithms: A_1, A_2 and A_3.

A_1:For each candidate c_i we calculate the area of $Vis(c_i)$. At each step we select the candidate whose visibility polygon has the smallest area.

A_2: For each candidate c_i we calculate $Vis(c_i)$ and determinate the number of intersections with the visibility polygons of the other candidates. At each step we select the candidate that has the smallest number of intersections.

A_3: For each candidate c_i we calculate the number of candidates it sees. At each step we select the one that sees the smallest number of points in *C*. This method is one of the methods developed in [4].

In all these algorithms, after adding a point to *S*, we remove from *C* all the candidates c_j such that $Vis(c_j)$ intersects the union of the visibility polygons of the elements on *S*. We stop when *C* is empty.

Algorithm A_1 is illustrated below.

Algorithm Determining S (greedy algorithm A_1)

Input: A polygon *Q* with *n* vertices
Output: A visibility-independent set *S*

1. $S \leftarrow \phi$
2. $C \leftarrow C_1 \cup C_2$
3. **for each** $c_i \in C$ **do**
4. determine the area of $Vis(c_i)$
5. **end for**
6. **while** $C \neq \phi$ **do**
7. choose the $c_i \in C$ whose $Vis(c_i)$ has the smallest area
8. $S \leftarrow S \cup \{c_i\}$
9. remove c_i from C and all c_j such that $Vis(c_j) \cap \bigcup_{s \in S} Vis(s) \neq \phi$
10. **end while**
11. **return** *S*

Algorithms A_2 and A_3 are very similar to this one, the main differences are in steps 4 and 7. It turns out that A_2 obtains the best results and A_3 the worst, both for orthogonal and arbitrary polygons.

5 Experiments and Results

According to section 3.1, we have several choices for two of the GA's parameters: selection and crossover. The different combinations produce eight methods:

- **Method 1** (M_1): Roulette Wheel Selection and Single Point Crossover
- **Method 2** (M_2): Roulette Wheel Selection and Two-Point Crossover
- **Method 3** (M_3): Roulette Wheel Selection and Uniform Crossover
- **Method 4** (M_4): Roulette Wheel Selection and a Variant of the Single Point Crossover
- **Method 5** (M_5): Tournament Selection and Single Point Crossover
- **Method 6** (M_6): Tournament Selection and Two-Point Crossover
- **Method 7** (M_7): Tournament Selection and Uniform Crossover
- **Method 8** (M_8): Tournament Selection and a Variant of the Single Point Crossover

The example given in Fig.8 illustrates an arbitrary polygon for which the visibility-independent set was obtained with A_2 and the solution obtained with the method M_8.

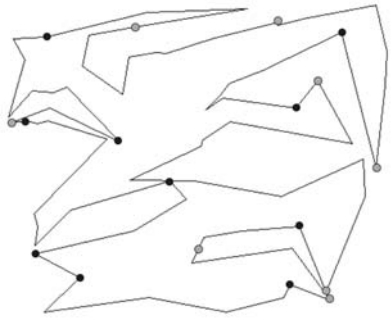

Fig.8: *S* and *G* sets (represented by gray and black dots, respectively) obtained with A_2 and M_8 on an arbitrary polygon with $n = 70$.

Fig.9 shows an orthogonal polygon for which the visibility-independent set was obtained with A_2 and the solution obtained with the method M_8.

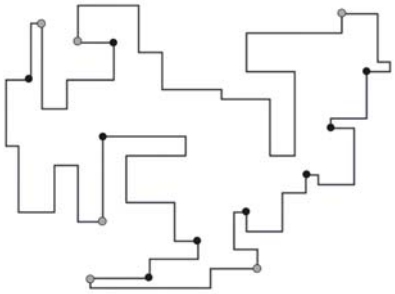

Fig.9: *S* and *G* sets (represented by gray and black dots, respectively) obtained with A_2 and M_8 on an orthogonal polygon with $n = 70$.

Fig.10 shows a comb polygon that we tested with M_8, and Figure Fig.11 shows the orthogonal version of a comb polygon that we tested with M_8. As is well known these polygons establish the necessity of $\lfloor n/3 \rfloor$ and $\lfloor n/4 \rfloor$ guards on arbitrary and orthogonal polygons, respectively [20]. Figures 10 and 11 also illustrate the visibility-independent set obtained by A_2, $|S| = 5$, and the solution obtained by M_8, $|G| = 5$ in both cases. It's important to observe that in these examples, $|G|/|S| = 1$, which means that our solution *G* is an optimal vertex-guard set in both cases.

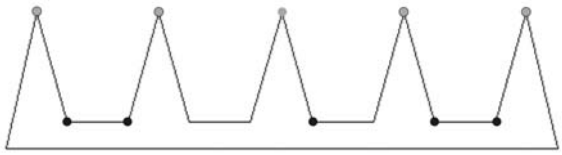

Fig.10: *S* and *G* sets (represented by gray and black dots, respectively) obtained with A_2 and M_8 in a comb polygon with $n = 15$.

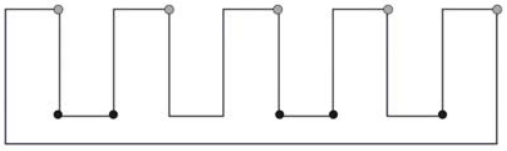

Fig.11: *S* and *G* sets (represented by gray and black dots, respectively) obtained with A_2 and M_8 in an orthogonal comb polygon with $n = 20$.

The implementation was done in C/C++ (for MS Visual Studio 2005) on top of the CGAL3.2.1. The above described methods were tested on a PC featuring a Intel(R) Core (TM)2 CPU 6400 at 2.66 GHz and 1 GB of RAM.

Our experiments were performed on a large set of randomly generated polygons (arbitrary and orthogonal). The arbitrary polygons were generated using the CGAL's function *random_polygon*_2 [12], whose implementation is based on the method of eliminating self-intersections in a polygon by using the so-called "2-opt" moves method; and to generate orthogonal polygons we used the polygon generator developed by Joseph O'Rourke (personal communication 2002). In the next two sections we will discuss the results of our experiments on arbitrary and orthogonal polygons, respectively.

5.1 Arbitrary Polygons

As stated above, we performed our experiments for arbitrary polygons on a large set of randomly generated polygons. We analyze the methods M_1, M_2, M_3, M_4, M_5, M_6, M_7 and M_8 by comparing the number of vertex-guards, the runtime and the number of iterations. The following experiments were done with four sets of arbitrary polygons, each with 40 polygons of 30, 50, 70 and 100 vertex polygons, respectively. In Tables 2, 3, 4 and 5 are exposed the results obtained by the first four methods. Note that, the tables show the average number of vertex-guards, the average runtime (in seconds) and the average number of iterations of the algorithm.

Number of vertices	$\lvert G \rvert$	Time (seconds)	Iterations
30	5.275	19.225	740.025
50	8.400	72.300	1134.300
70	11.725	190.050	1763.200
100	17.000	504.525	2690.300

Table 2: Results obtained with M_1.

Number of vertices	$\lvert G \rvert$	Time (seconds)	Iterations
30	5.350	18.875	709.750
50	8.400	72.175	1130.400
70	11.675	177.050	1632.400
100	16.925	468.550	2532.400

Table 3: Results obtained with M_2.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	5.275	17.625	685.225
50	8.275	68.825	1098.900
70	11.575	171.300	1641.600
100	16.900	460.450	2604.400

Table 4: Results obtained with $M3$.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	5.225	10.700	721.200
50	8.250	60.350	1145.000
70	11.750	158.500	1632.200
100	16.775	443.950	2529.500

Table 5: Results obtained with M_4.

As we can see, in these first four methods there are almost no differences on the average number of vertex-guards obtained. Though M_4 seems to be the one that obtains slightly better solutions, except for $n=70$. Concerning the average runtime, M_4 seems to be the best one.

In the following four cases, we analyze how the different types of crossover behave, considering the tournament selection. The obtained results are shown in Tables 6, 7, 8 e 9.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	5.350	17.325	686.200
50	8.500	60.900	1009.700
70	12.000	142.975	1399.600
100	16.850	384.775	2173.100

Table 6: Results obtained with M_5.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	5.350	16.800	682.875
50	8.450	59.300	973.175
70	11.850	138.525	1355.800
100	17.000	365.600	1033.400

Table 7: Results obtained with M_6.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	5.300	16.725	680.425
50	8.475	57.400	958.100
70	11.775	130.125	1296.800
100	16.825	321.325	1822.500

Table 8: Results obtained with M_7.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	5.200	9.050	709.425
50	8.375	44.925	985.800
70	11.625	110.975	1324.700
100	16.900	310.500	1987.700

Table 9: Results obtained with M_8.

Again, in these four methods there are almost no differences on the average number of vertex-guards obtained. However, M_8 seems to be the method that obtains slightly better solutions, with the exception of $n=100$. Concerning the average runtime, M_8 also seems to be the best one.

Comparing the eight methods, we notice that the obtained results, concerning the average of $|G|$, are approximately the same for all methods. However, M_4 and M_8 are the methods that seem to achieve slightly better solutions. Concerning the average runtime, M_8 seems to be the best method.

Nevertheless, the comparison between the obtained data by our eight methods only makes sense if a statistical study is made to ensure the statistically significance of the results [2]. So, first of all, we studied the results related to the number of vertex-guards. To check the data normality we applied the Kolmogorov-Smirnof test (using the Statistics Toolbox (Version 7.3) of the MATLAB software), and we obtained the following *p*-values for the method M_1: $1.3471e^{-036}$, for $n=30$ and $1.0754e^{-036}$, for $n=50,70$ and 100. These values mean that there is always a case that the data is non-normally distributed. So, we used the Kruskal-Wallis test to compare our data. In this test we declare that a result is significantly different if the *p*-value is less than 0.05. The *p*-values returned by the Kruskal-Wallis test are 0.9676, 0.9565, 0.9191 and 0.9933 for the data obtained with the polygons with $n=30,50,70$ and 100, respectively. So we can say that there are no significantly differences between the eight methods, regarding $|G|$.

According to the previous conclusion, we continued our statistical study regarding the runtime. To check the data normality we applied the Kolmogorov-Smirnof test, and we obtained the following *p*-values for the method M_1: $1.0754e^{-036}$, for $n=30,50,70$ and 100. Consequently, we used again the Kruskal-Wallis test to compare our data. The *p*-values returned by the Kruskal-Wallis test are 0, for the data obtained with the polygons with $n=30,50,70$ and 100. So we can conclude that at

least one method is significantly different, concerning the runtime. Then we performed a multiple comparison test to determine which pairs of methods are significantly different, and which are not (using the MATLAB's multicompare function). The results provided by the tests allow us to conclude that M_8 is always the best method. As we want to find a compromise between the goodness of the solution and the algorithm's runtime we continue our study considering that M_8 is the algorithm that obtains the best results.

It is important to note the infinity of the alternatives left to explore with respect to parameters of the GA metaheuristic. In this work, we attempt to calculate references to these parameters, noting that a more exhaustive study in future investigations might improve the obtained results.

Now, to infer about the average of the minimum number of vertex-guards needed to cover an arbitrary polygon with *n* vertices, we applied M_8 to eight sets of orthogonal polygons, each one with 40 polygons of 30, 50, 70, 100, 110, 130, 150 and 200 vertex polygons, respectively. The average of the obtained results, concerning $|G|$, are exposed in Table 10.

Number of vertices	$\|G\|$
30	5.200
50	8.375
70	11.625
100	16.900
110	18.250
130	21.800
150	25.225
200	31.075

Table 10

Using the least squares method, we obtained the following linear adjustment (see Fig.12):

$$f(x) = 0.1566x + 0.8684 \approx \frac{x}{6.38} + 0.8684 \approx \frac{x}{6.38}$$

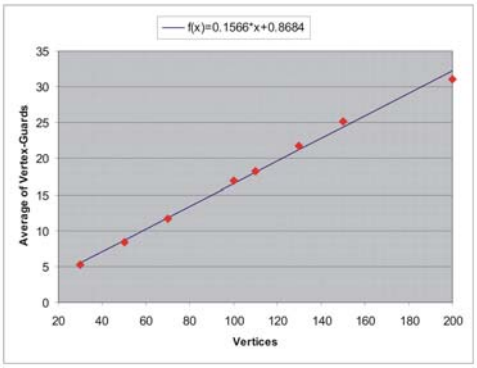

Fig.12: Average number of guards for arbitrary polygons

Hence, we can conclude that, on average, the minimum number of vertex-guards needed to cover an arbitrary polygon with *n* vertices is $\frac{n}{6.38}$. This is a much better result than the known theoretical upper bound $\lfloor n/3 \rfloor$. In order to get a quantitative measure on the quality of the calculated $|G|$, we computed the visibility-independent sets for our instances (the eight sets of polygons described above). The ratio between the smaller *G* (obtained with M_8) and the larger visibility-independent set, *S* obtained with A_2 (see section 4) never exceeded 2. That implies that our algorithm has an approximation ratio of 2. It should be noted, however, that the approximation ratio in most cases is less than 2, with an average of 1.3 for the 320 polygons.

5.2 Orthogonal Polygons

A similar study was made with randomly generated orthogonal polygons.

We performed our experiments with four sets of orthogonal polygons, each with 40 polygons of 30, 50, 70 and 100 vertex polygons, respectively. The results obtained by the eight methods are shown in Tables 11, 12, 13, 14, 15, 16, 17 and 18.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.725	14.950	746.025
50	7.925	59.750	1150.900
70	10.675	162.000	1789.500
100	15.475	421.800	2738.500

Table 11: Results obtained with M_1.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.625	15.250	749.225
50	7.875	59.100	1151.200
70	10.850	157.575	1712.500
100	15.500	401.875	2601.800

Table 12: Results obtained with M_2.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.625	14.125	724.425
50	7.775	54.800	1102.700
70	10.325	149.875	1729.800
100	14.925	398.625	2745.400

Table 13: Results obtained with M_3.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.575	9.525	735.200
50	7.775	50.250	1148.100
70	10.600	145.675	1733.800
100	15.350	387.425	2673.100

Table 14: Results obtained with M_4.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.700	14.050	712.850
50	7.900	50.050	1035.000
70	10.725	124.450	1399.800
100	15.700	312.400	2058.200

Table 15: Results obtained with M_5.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.750	14.400	740.175
50	7.900	48.725	985.700
70	10.725	122.075	1394.200
100	15.550	296.850	1964.000

Table 16: Results obtained with M_6.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.700	12.800	681.425
50	7.900	44.975	932.650
70	10.850	110.700	1269.500
100	15.025	285.800	1994.400

Table 17: Results obtained with M_7.

Number of vertices	$\|G\|$	Time (seconds)	Iterations
30	4.750	7.100	684.475
50	7.825	37.525	995.625
70	10.675	99.475	1339.800
100	15.375	265.625	1973.400

Table 18: Results obtained with M_8.

Comparing the eight methods, we notice that the obtained results, concerning the average of $|G|$, are approximately the same for all methods; and M_3 is the method that seems to achieve a slightly better solution. Concerning the average runtime, M_8 seems to be the best one. For the same reason as before, concerning arbitrary polygons, we analyzed our results using a statistical study. We concluded that M_8 is, again, the algorithm that achieves the best results. Then we applied this algorithm to eight sets of orthogonal polygons, each one with 40 polygons of 30, 50, 70, 100, 110, 130, 150 and 200 vertex polygons, respectively (see Table 19, for the average of the obtained results, concerning $|G|$).

Number of vertices	$\|G\|$
30	4.750
50	7.825
70	10.675
100	15.375
110	17.200
130	20.250
150	23.375
200	31.200

Table 19

Then, using the least squares method, we get the next linear adjustment (see Fig. 13):

$$f(x) = 0.1561x - 0.0556 \approx \frac{x}{6.40} - 0.0556 \approx \frac{x}{6.40}$$

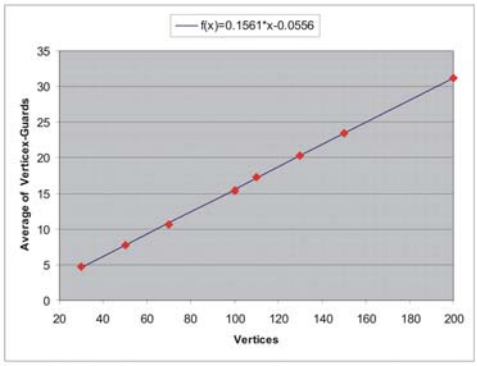

Fig.13: Average number of guards for orthogonal polygons.

So, we conclude that, on average, the minimum number of vertex-guards needed to cover an orthogonal polygon with n vertices is $\frac{n}{6.40}$, which improves the theoretical upper bound $\lfloor n/4 \rfloor$. And the approximation ratio of M_8 is 1.9 (being, in most cases less than 1.9, with an average of 1.4 for the 320 polygons).

6 Conclusions and Future Work

We designed and implemented eight approximation algorithms to solve the MINIMUM VERTEX GUARD problem on polygons based on general metaheuristic genetic algorithms. We conducted an experimental study of their performance on polygons, arbitrary and orthogonal, and we made a statistical study to elect the best method. Then, using the elected algorithm we concluded that, on average, the minimum number of vertex-guards on a polygon with n vertices is $\frac{n}{6.38}$, both for arbitrary polygons and $\frac{n}{6.40}$ for orthogonal polygons. The approximation ratio of the best algorithm is equal to 2, for arbitrary polygons and is equal to 1.9, for orthogonal polygons.

Since the proposed methods, based on metaheuristic approach - Genetic Algorithm, have proven to behave well to solving the MINIMUM VERTEX GUARD problem on polygons, there are several directions for further research. We intend to adapt and implement other metaheuristics (e.g., Simulated Annealing) and to develop *hybrid metaheuristics* to try and improve the solution of this problem, as well as to solve other NP-hard visibility problems.

References:

[1] E. Alba. Parallel Metaheuristics: a New Class of Algorithms, Wiley-Interscience Publishers, 2005.

[2] E. Alba and G. Luque, Measuring the Performance of Parallel Metaheuristics, In: E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley-Interscience Publishers, 2005.

[3] M. Abellanas, E. Alba, S. Canales and G. Hernández. Resolución de un problema de iluminación con Simulated Annealing (in spanish), Actas de MAEB'07, Tenerife, España, 2007.

[4] Y. Amit and J. S. B. Mitchell and E. Packer. Locating Guards for Visibility Coverage of Polygons, Proceedings of the Workshop on Algorithm Engineering and Experiments, 1-15, 2007.

[5] A.L. Bajuelos, S. Canales, G. Hernández and A.M. Martins. Solving some combinatorial problems in grid *n*-ogons, in Proceedings of the 7th International Conference on Applied Computer Science (ACS'07), Volume 7, 151-156, Venice, Italy, 2007.

[6] A.L. Bajuelos, S. Canales, G. Hernández, A.M. Martins. Estimating the Maximum Hidden Vertex Set in Polygons, Proceedings of ICCSA'08, 421-432, IEEE-CS Press, Perugia, Italy, 2008.

[7] C. Blum and R. Andreia. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computers. Survey, 35(3), 268-308, September, 2007.

[8] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org

[9] V. Chvátal. A combinatorial theorem in plane geometry, J. of Combinatorial Theory (Series B) 18 (1975) 39-41.

[10] S.K. Ghosh. Approximation Algorithms for Art Gallery Problems, Proceedings of the Canadian Information Processing Society Congress, 429-434, 1987.

[11] F. Glover and G.A. Kochenberger. Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, 2003.

[12] S. Hert, M. Hoffmann, L. Kettner, and S. Schonherr. Geometric object generators. In C. E. Board, editor, CGAL User and Reference Manual. 3.2.1 edition, 2006.

[13] J. Kahn, M. Klawe, D. Kleitman. Traditional galleries require fewer watchmen. SIAM J. Algebraic and Discrete Methods 4 (1983) 194-206.

[14] D. Lee and A. Lin. Computational Complexity of Art Gallery Problems, IEEE Transactions on Information Theory IT-32, 276-282, 1996.

[15] C. Marcelo, C. C. de Souza, P.J. Rezende. An Exact and Efficient Algorithm for the Orthogonal Art Gallery Problem, Proceedings of XX Brazilian Symposium on Computer Graphics and Image Processing, v. 1, 87-94, 2007.

[16] M. Maric, M. Tuba, J. Kratica. One Genetic Algorithm for Hierarchical Covering Location Problem, in Proceedings of the 9th WSEAS International Conference on Evolutionary Computing (EC'08), 122-126, Sofia, Bulgaria, 2008.

[17] C.R. Reeves. Genetic Algorithms, In: Handbook of Metaheuristics, F. Glover e G.A. Kochenberger (eds). Kluwer, Boston, 55-82, 2003.

[18] D. Schuchardt and H. Hecker. Two NP-Hard Art-Gallery Problems for Ortho-Polygons. Math. Logic Quarterly, 41(2),261-267, 1995.

[19] A.P. Tomás, A. L. Bajuelos, F. Marques. Approximation Algorithms to Minimum Vertex Cover Problems on Polygons and Terrains, LNCS 2657, Springer-Verlag, 869-878, 2003.

[20] J. Urrutia. Art Gallery and Illumination Problems. In J.-R. Sack and J. Urrutia (eds) Handbook of Computational Geometry, Elsevier, 2000.

[21] K.S. Yildirim, T.E. Kalayci, A. Ugur. Optimizing Coverage in a K-Covered and Connected Sensor Network Using Genetic Algorithms, in Proceedings of the 9th WSEAS International Conference on Evolutionary Computing (EC'08), 21-26, Sofia, Bulgaria, 2008.